Traitement des signaux audio : principes et expérimentations

S. Lesoinne et J.J. Embrechts

5 février 2007

Table des matières

Première partie

INTRODUCTION

Chapitre 1

Les DSPs

1.1 DSP et micro-processeurs

Un DSP est un type particulier de micro-processeur. La différence principale entre les deux réside dans le fait que contrairement au micro-processeur qui n'est pas destiné à une application spécifique, l'architecture, ses instructions et l'ensemble de ses fonctionnalités ont été choisies afin de le rendre particulièrement performant dans le domaine du traitement du signal. Comme un micro-processeur classique, on peut lui adjoindre de la mémoire (RAM, ROM), et des périphériques. Il se présente généralement sous la forme d'un micro-contrôleur intégrant

- de la mémoire,
- des timers,
- des ports séries synchrones rapides,
- des contrôleurs DMA et,
- des ports d'entrées/sorties divers.

1.2 Grandes familles de DSPs

Le marché est partagé entre quatre constructeurs principaux : Texas Instruments, Analog Devices, Freescale (Motorola) et Lucent. Les DSP se différencient par le format de calcul (fixe ou entier), la taille du bus de données (16, 24 ou 32 bits), la puissance en millions d'instructions par secondes (MIPS) et les fonctionalités spécifiques directement intégrées (traitement du son, de l'image, etc.).

Les processeurs **à virgule fixe** lisent les bits comme des fractions en puissances négatives de 2. Les nombres représentables sont compris entre -1 et 1-ϵ. L'architecture de calcul en est simplifée et son coût est moindre.

Les processeurs **à virgule flottante** utilisent une représentation des nombres sous forme d'exposant et de mantisse, ce qui implique une architecture plus complexe et un coût plus élevé.

Chapitre 2

Rappels

2.1 Traitement du signal

2.1.1 Filtres (audio) numériques

Un filtre numérique est un système de traitement numérique qui agit sur un signal numérique d'entrée $x[n]$ et produit un autre signal numérique à sa sortie $y[n]$.



FIG. 2.1 – Filtre numérique

Causalité : si

$$x[n] = 0\,, \forall n < n_0$$

alors

$$y[n] = 0\,, \forall n < n_0$$

Systèmes linéaires invariants : La plupart des systèmes peuvent se mettre sous la forme d'une équation aux différences linéaires à coefficients constants

$$b_0 y[n] + ... + b_N y[n-N] = a_0 x[n] + \ldots + a_M x[n-M]$$

Réponse impulsionnelle (principe de superposition) : La réponse d'un système LTI est de la forme

$$y[n] = \sum_{l=-\infty}^{+\infty} x[l].g[n-l]$$

ou

$$y[n] = x[n] \star g[n]$$

où $g[n-l]$ est la réponse à l'impulsion unité $d[n-l]$.

Si le système est, de plus, causal et $x[n] = 0, \forall n < 0$ alors, la réponse impulsionnelle devient

$$y[n] = \sum_{l=0}^{+\infty} x[l].g[n-l]$$

Stabilité : Un système LTI est stable si $\sum_{k=-\infty}^{+\infty} |g[k]| < \infty$.

2.1.2 Transformée en Z

$$X(z) = \sum_{n=-\infty}^{\infty} x[n].z^{-n} \text{ , } z \, complexe$$

Propriétés :
- $x[n] <\text{———}> X(z)$
- $x[n-n_0] <\text{———}> z^{-n_0}.X(z)$
- $x[n] \star y[n] <\text{———}> X(z).Y(z)$

2.1.2.1 Fonction de transfert

La transformée en Z d'un système LTI dont l'équation aux différences est du type

$$b_0 y[n] + b_1 y[n-1] + \ldots + b_N y[n-N] = a_0 x[n] + a_1 x[n-1] + \ldots + a_M x[n-M]$$

est de la forme

$$G(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1.z^{-1} + \ldots + a_M.z^M}{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}} \quad (2.1)$$

2.1.2.2 Stabilité

Un système de la forme (eq. 2.1) est stable et causal si tous ses pôles sont tels que

$$|z| < 1$$

2.1.2.3 Système à minimum de phase

Un système de la forme (eq. 2.1) est dit à minimum de phase si tous ses zéros sont tels que

$$|z| < 1$$

2.1.3 Filtres non récursifs et filtres FIR

La sortie d'un filtre non récursif ne fait intervenir que les échantillons précédents de l'entrée

$$y[n] = \frac{a_0}{b_0} x[n] + \ldots + \frac{a_M}{b_0} x[n-M]$$

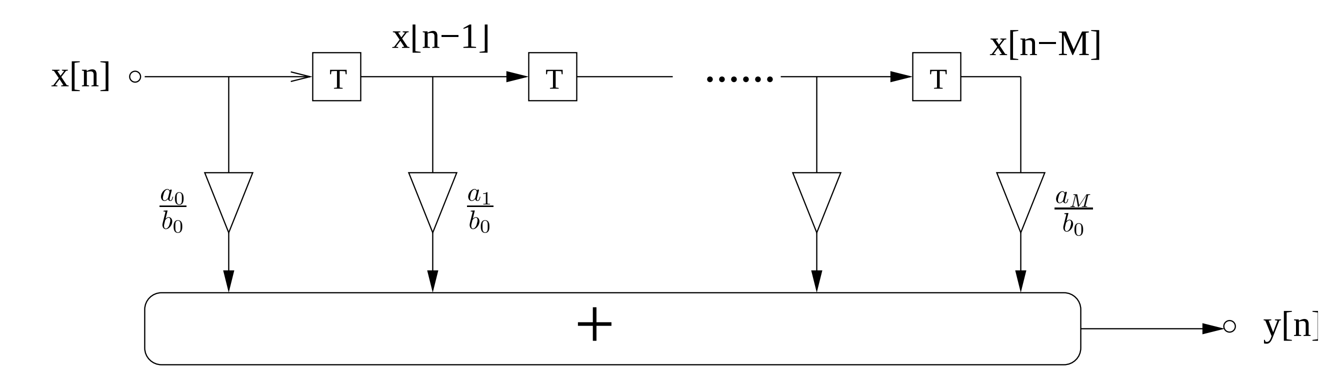


FIG. 2.2 – Filtre non récursif

Un filtre non récursif est toujours à réponse impulsionnelle finie (FIR), mais pas l'inverse !

2.1.4 Filtres récursifs et filtres IIR

La sortie d'un filtre récursif fait intervenir tant les échantillons précédents de l'entrée que ceux de la sortie, i.e. $b_1, \ldots, b_N \neq 0$.

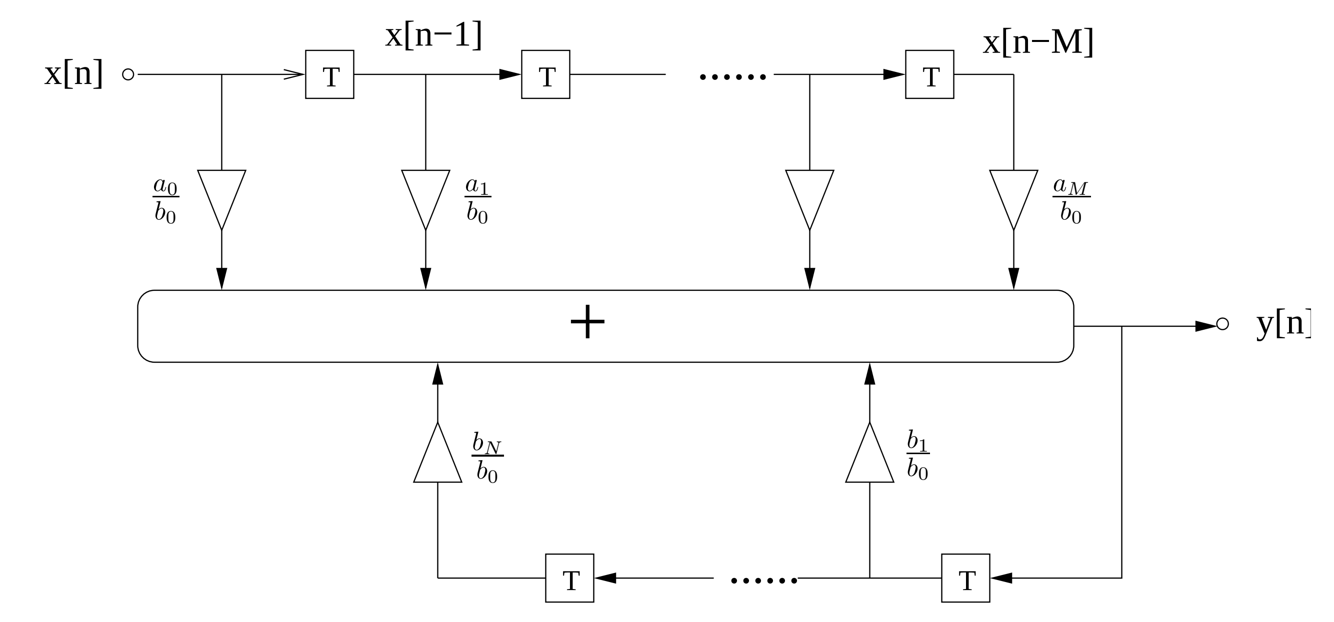


FIG. 2.3 – Filtre récursif

Un filtre à réponse impulsionnelle infine (IIR) est toujours récursif, mais pas l'inverse!

2.2 Calculs en virgule fixe

2.2.1 Notation

Le code binaire naturel ou absolu permet de représenter un nombre entier E sur N bits b_n selon l'équation suivante :

$$E = \sum_{n=0}^{N-1} b_n.2^n = 2^{N-1}.b_{N-1} + 2^{N-2}.b_{N-2} + \ldots + 2^1.b_1 + 2^0.b_0$$

2.2.2 Le code complément à 2

Ce code permet de représenter en binaire les nombres **entiers** positifs et négatifs. En utilisant N bits, on code les nombres entiers positifs avec les 2^{N-1}premières combinaisons ($b_{N-1} = 0$) en utilisant les conventions des entiers non signés et les entiers négatifs avec les 2^{N-1}combinaisons restantes en utilisantle nombre qu'il faudrait ajouter au module pour obtenir 0. Donc, le bit le plus significatif est à 1 dans le cas d'un nombre négatif et à 0 sinon.

Un nombre entier signé E peut encore s'écrire

$$E = -2^{N-1}.b_{N-1} + 2^{N-2}.b_{N-2} + \ldots + 2^1.b_1 + 2^0.b_0$$

Pour former un nombre négatif à partir d'un nombre positif, il suffit d'inverser tous les bits et d'additionner 1.

Exemple

On peut obtenir -72 à partir de 72 de la façon suivante

$Code\, binaire\, de\, 72$	0100 1000
$Inversion\, des\, bits$	1011 0111
$Additionner\, 1$	0000 0001
$Code\, de\, -72$	1011 1000

Suggestion :

1. vérifiez que l'addition binaire de -72 et de son module donne zéro
2. faites la même vérification pour un nombre négatif quelconque (E)

2.2.3 Les nombres fractionnaires à virgule fixe

Jusqu'à présent, la discussion s'est concentrée sur la représentation binaire des entiers. Il est cependant tout aussi utile de pouvoir représenter des **fractions** en **binaire**[1].

Dans les formats considérés jusqu'à présent, on a supposé que le radix (l'équivalent en binaire du point en décimal) était à droite de l'entier binaire. En utilisant les fractions binaires, le radix se déplace de n bits vers la gauche en fonction de la représentation choisie.

- Le format Q_n, couramment adopté, spécifie qu'un nombre binaire possède n bits à droite du radix.
- n+1 bits sont nécessaires pour contenir un nombre au format Q_n.
- Les nombres négatifs sont représentés en complément à 2.
- Les nombres fractionnaires sont compris entre $-1.0\ldots$ et $+0.9999$.

Par exemple, en représentation Q_7, quelle est la valeur décimale de 10111101 ?

C'est un nombre négatif, il faut donc le convertir en nombre non signé, ce qui nous donne : 01000011. Vient ensuite la conversion en décimal :

$$0.sgn + 1.\frac{1}{2} + 0.\frac{1}{4} + 0.\frac{1}{8} + 0.\frac{1}{16} + 0.\frac{1}{32} + 1.\frac{1}{64} + 1.\frac{1}{128} = 0.5234375$$

et puisque le nombre était négatif, le résultat final vaut -0.5234375.

Formulation

Cette conversion s'effectue selon la formule suivante

$$F = -2^0.b_{N-1} + 2^{-1}.b_{N-2} + \ldots + 2^{-(N-2)}.b_1 + 2^{-(N-1)}.b_0$$

où b_{N-1}est le bit de poids fort qui correspond aussi au bit de signe et $b_0$est le bit de poids faible.

Remarque

Puisque tout nombre réel ne peut être représenté par un nombre binaire de taille fixe qu'avec une précision limitée, il convient d'être très prudent quand une solution théorique est implémentée sur DSP. Le cas se présente notamment pour les coefficients de filtres.

[1] L'un des avantages est que lors de la multiplication de deux fractions de module < 1, le résultat restera inférieur à 1 en valeur absolue.

Deuxième partie

DSPs Motorola de la famille 56300

Chapitre 3

Architecture interne

3.1 Architecture Harvard

Avec une architecture Von Neumann classique, le CPU peut soit lire une instruction, soit lire/écrire une donnée de/vers la mémoire mais, jamais les deux simultanément car les instructions et les données partagent un même bus et une même mémoire.

Par contre, l'architecture Harvard, contrairement à l'architecture Von Neumann, utilise deux bus pour les accès mémoire : un pour le transfert de données et l'autre pour les instructions. Il existe une variante à cette architecture appelée Harvard double qui utilise trois bus répartis comme suit : un pour les instructions et deux pour les données, ce qui permet l'exécution d'une instruction en parallèle avec des accès mémoire.

3.2 Schéma général

Les DSPs de la famille 56300 s'articulent autour d'un noyau à virgule fixe de 24 bits auquel on peut ajouter divers périphériques standards (port série, port parallèle, ports entrées/sorties généralistes, timers, modules mémoire RAM/ROM, coprocesseurs divers) tel qu'illustré à la figure 3.1.

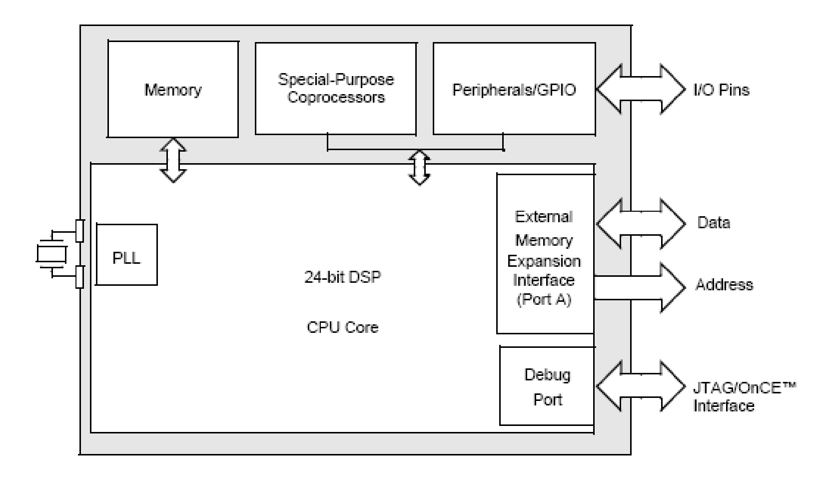


FIG. 3.1 – Structure de la famille 56300 (source : manuel de la famille 56300)

Comme on peut l'observer à la figure 3.2 pour le cas particulier du DSP 56311,

le noyau se compose, entre autres, d'une Data Arithmetic Logic Unit (Data ALU), d'une Adress Generation Unit (AGU), d'une Program Control Unit (PCU), d'une Phase Locked Loop (PLL), d'une interface pour la mémoire externe (Port A), d'un Instruction cache controler embarqué, d'un support matériel de débuggage (JTAG Test Access Port,*OnCE*™), d'un controleur DMA (en dessous de l'AGU).

On y distingue différentes zones :

- zone de périphériques d'expansion
- zone comprenant les modules mémoires internes
- les différents bus, tous de taille 24 bits (y compris les bus d'adresses)
- modules de gestion des périphériques externes et de débuggage
- modules de contrôle de l'exécution du programme
- la Data ALU
- zone de gestion mémoire (AGU, DMA, sélection d'adresses externes).

Les bus de données, au nombre de deux (Harvard double), sont notés XDB (X Data Bus) et YDB (Y Data Bus).

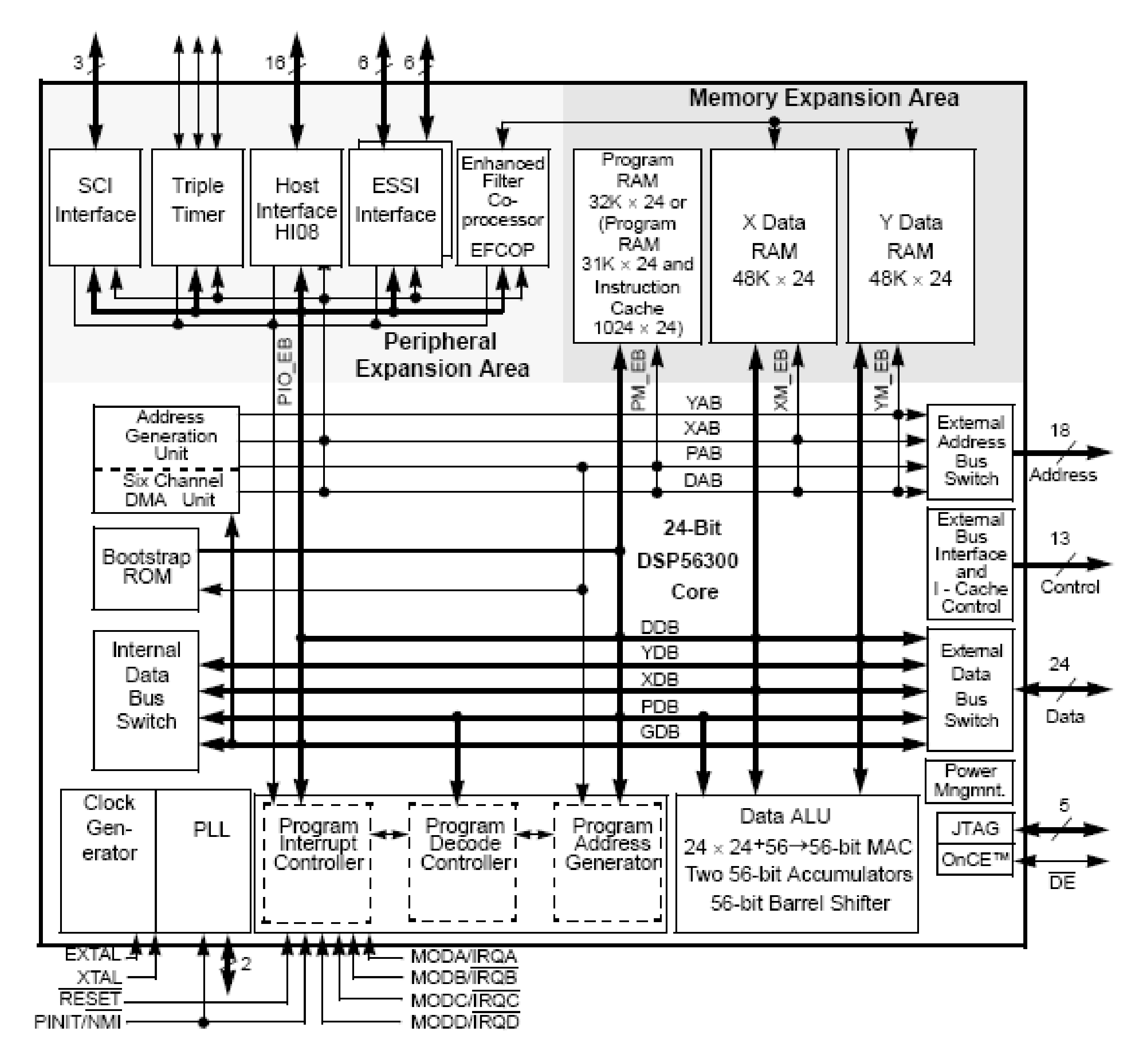


FIG. 3.2 – Schéma bloc du 56311 (source : manuel d'utilisateur du DSP 56311)

La fonction des autres bus est reprise à la figure 3.3

Global Data Bus	GBD	Between Program Control Unit and other core structures
Peripheral I/O Expansion Bus	PIO_EB	To peripherals
Program Memory Expansion Bus	PM_EB	To Program ROM
Program Data Bus	PDB	Carries program data throughout the core
Program Address Bus	PAB	Carries program memory addresses throughout the core
X Memory Expansion Bus	XM_EB	To X memory
X Memory Data Bus	XDB	Carries X data throughout the core
X Memory Address Bus	XAB	Carries X memory addresses throughout the core
Y Memory Expansion Bus	YM_EB	To Y Memory
Y Memory Data Bus	YDB	Carries Y data throughout the core
Y Memory Address Bus	YAB	Carries Y memory addresses throughout the core
DMA Data Bus	DDB	Transfers data with DMA channels
DMA Address Bus	DAB	Transfers address information with DMA channels

FIG. 3.3 – Fonction des bus apparaissant à la figure 3.2 (source : manuel d'utilisateur du DSP 56311).

3.3 Data ALU

La Data ALU effectue toutes les opérations logiques et arithmétiques sur les données. Elle est principalement composée (cfr. figure 3.4) de :
- 4 registres d'entrée 24 bits X0, X1, Y0, Y1.
- une unité MAC (Multiplier-Accumulator) avec pipeline
- 2 accumulateurs de 56 bits chacun A et B (décomposés en 2 registres de 48 bits et 2 registres d'extension de 8 bits)
- une unité de décalage pour l'accumulateur (accumulator shifter)
- des circuits de décalage et limitation pour les deux bus de données.
- une BFU (Bit Field Unit).

3.3.1 Les registres d'entrée

Les registres de la Data ALU peuvent être traités comme des opérandes de 24 ou 48 bits, acquises ou lues sur les bus de données X (XDB) ou Y (YDB). Les opérandes sources pour la Data ALU (qui peuvent être de 24, 48 ou 56 bits) ne peuvent provenir que de ses registres (accumulateurs compris) à une exception : lorsque la source est contenue dans l'instruction (immediate field), elle provient du PDB et passe par le multiplexeur. Le résultat de toute opération de la Data ALU est stocké dans un accumulateur (A ou B).
- Les registres d'entrée X0, X1, Y0, Y1 servent de buffers d'entrée entre les bus XDB, YDB et l'unité MAC ou l'unité à décalage multiple (barrel shifter). Ils peuvent être traités comme quatre registres indépendants ou deux registres de 48 bits, X et Y, formés par la concaténation de X1 :X0[1] et Y1 :Y0 respectivement.

[1]La notation X1 :X0 représente la concaténation de X1 et X0 où X1 fournit les bits les plus significatifs (resp. X0 fournit les bits les moins significatifs)

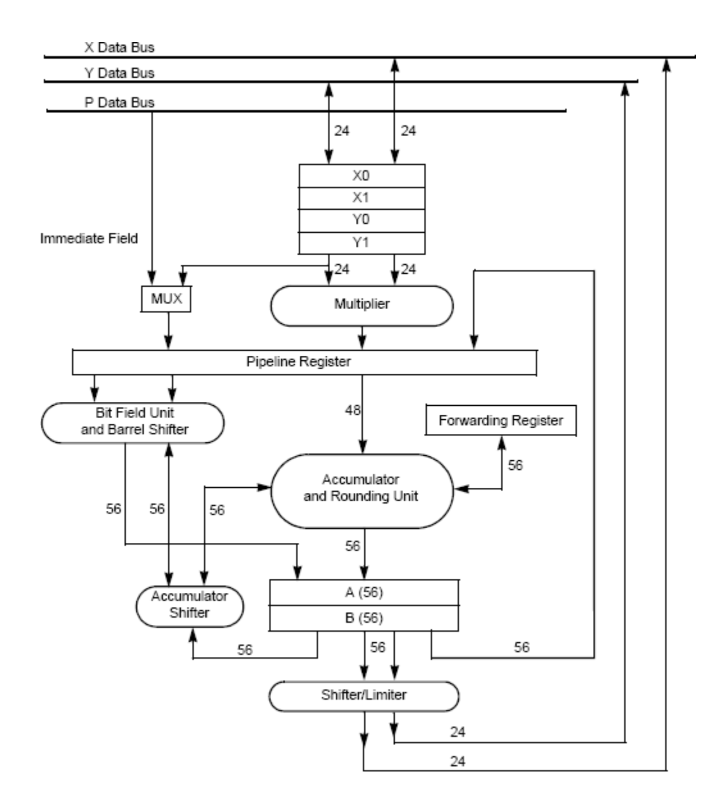


FIG. 3.4 – Schéma bloc de la Data ALU (source : manuel de la famille 56300)

3.3.2 L'unité MAC

L'unité Multiplicateur-Accumulateur (MAC) est l'unité principale de traitement arithmétique du noyau du DSP 56300. Le multiplicateur exécute des multiplications 24bits X 24bits et le résultat de 48 bits est justifié à droite en 56 bits et additionné au contenu de l'accumulteur A ou B. Quand un résultat de 56 bits doit être stocké en tant qu'opérande 24 bits, soit la partie la moins significative est tronquée, soit le résultat de 56 bits est arrondi à 24 bits, en fonction de l'instruction DSP en cours d'exécution. Le type d'arrondi est spécifié par le bit correspondant dans un registre spécial : le registre d'état SR (Status Register).

3.3.3 Les accumulateurs

Les accumulateurs A et B sont constitués par la concaténation de 3 registres chacun (A2 :A1 :A0 et B2 :B1 :B0 respectivement). Le résultat d'une opération est stocké dans l'accumulateur comme suit :

- Dans A1 : les 24 bits les plus significatifs du produit (MSP)
- Dans A0 : les 24 bits les moins significatifs du produit (LSP)
- Dans A2 : les 8 bits d'extension (EXT).

Si le résultat d'une opération d'accumulation (addition) conduit à un dépassement des valeurs [-1, 1-ε] (overflow), les bits excédentaires seront stockés dans A2 (B2 resp.). Quand un accumulateur qui contient des bits excédentaires dans A2 (B2 resp.) est lu et sa valeur placée sur un bus de donnée (la taille de XDB et YDB est de 24 bits), la valeur effectivement placée sur le bus sera une constante de limitation. La valeur présente dans A (B resp.) n'est pas affectée. La constante de limitation est la valeur maximale (en valeur absolue) <u>de même signe</u> que la valeur présente dans l'accumulateur, c'est-à-dire soit -1, soit $1-\varepsilon$. Quand l'opération de limitation se produit, un drapeau est mis à "1" dans SR.

Si le résultat d'une opération ne conduit pas à un dépassement, les 8 bits d'extension prennent la valeur du bit de signe du résultat : on procède à une extension de signe.

Par exemple, si on utilise 6 bits pour représenter +13 (001101) et que l'opération d'extension de signe conduit à un résultat codé sur 10 bits (4 bits d'extension), la nouvelle représentation de +13 est obtenue en ajoutant quatre "0" aux 6 bits précédents : 0000 001101. Si le nombre de départ est négatif (-13 en représentation complément à 2 sur 6 bits = 110011), le résultat de l'extension de signe sur 10 bits est obtenu en ajoutant quatre "1" aux 6 bits précédant (1111 110011). De cette façon, la positivité ou négativité du nombre original est préservée.

Si un résultat de 24 bits doit être écrit dans un accumulateur, les 24 bits sont placés dans A1, tandis qu'A0 est rempli de "0" et A2 contient l'extension de signe.

3.4 AGU

L'AGU (unité de génération d'adresse) effectue les opérations de calcul (en arithmétique entière) des adresses effectives nécessaires pour accéder aux opérandes stockées en mémoire et contient les registres utilisés lors de ces calculs. Pour minimiser les surcoûts engendrés par ces calculs, l'AGU travaille en parallèle avec les autres unités de la puce. Elle traite quatre types d'adressage :

- linéaire
- modulo
- modulo à enroulements multiples
- reverse-carry (utilisé pour les FFTs).

En se rapportant à la figure 3.5, on observe qu'elle est scindée en deux parties comportant chacune une Address ALU et quatre triplets de registres :

- Rn : le registre d'adresse (Address Register) contient l'adresse à utiliser
- Mn : le registre d'index (Modifier Register) spécifie le type d'arithmétique à utiliser pour la mise à jour du registre d'adresse
- Nn : le registre d'offset (Offset Register) contient l'offset (décalage) éventuel à appliquer à Rn.

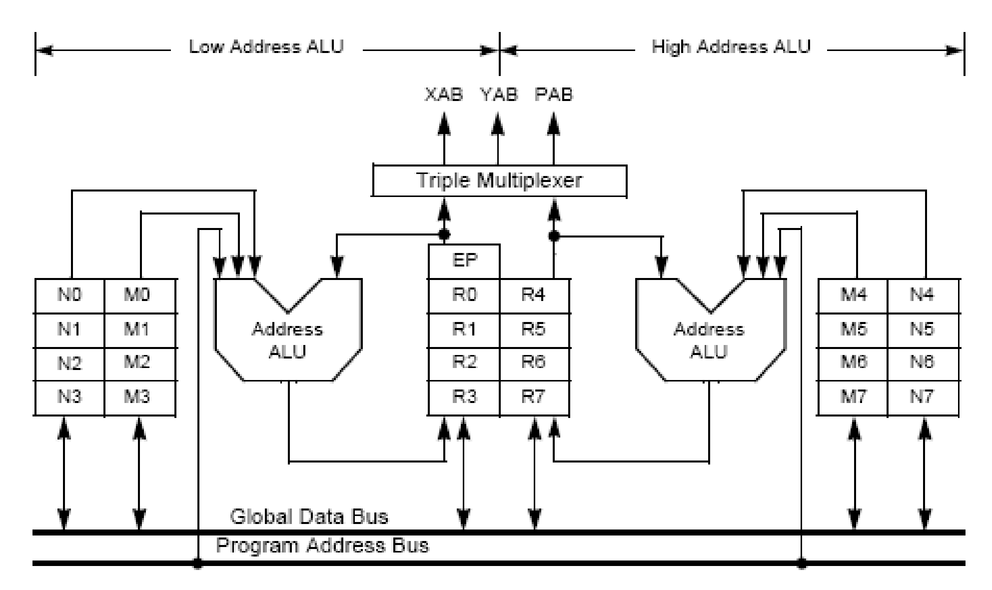


FIG. 3.5 – Schéma bloc de l'AGU (source : manuel de la famille 56300)

A chaque registre d'adresse Rn est associé un registre Mn et un registre Nn. Par exemple, pour le registre R_2, ce sont les registres M_2 et N_2 qui lui sont associés.

Chaque Address ALU peut travailler en concurrence avec l'autre, ce qui permet le calcul de deux adresses (bus XAB et YAB) en un cycle d'instruction.

Les bus PAB, XAB, YAB correspondent au bus de programme et aux bus d'adresse X, Y.

3.5 PCU

L'unité de contrôle du programme (PCU) coordonne l'exécution des instructions du programme et les instructions de traitement des interruptions et exceptions. Elle fonctionne à travers un pipeline d'instruction à 7 étages (présenté plus en détails dans la section 3.6) et plusieurs registres programmables. Elle est composée de trois blocs matériel (fig. 3.6) : le générateur d'adresse de programme (PAG), le contrôleur de décodage du programme (PDC), le contrôleur d'interruption du programme (PIC).

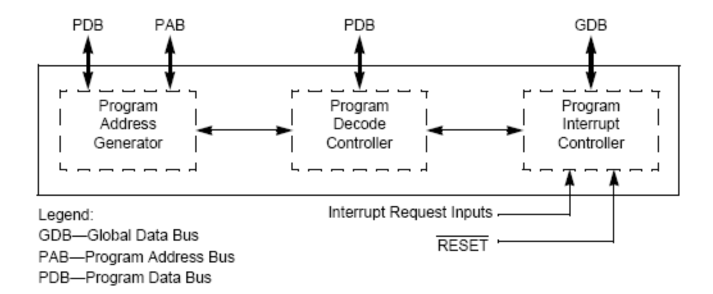


FIG. 3.6 – Architecture PCU (source : manuel de la famille 56300)

Pour préserver l'opération en cours et les valeurs de status pendant l'exécution des interruptions et des exceptions, la PCU dispose d'une pile système (system stack). De plus, elle fournit un support spécial pour les boucles (DO) et l'instruction de répétition (REPEAT). Pour gérer ses différents fonctions, elle s'appuie sur plusieurs registres programmables dont, entre autres, le registre du mode d'opération (OMR : operating mode register), le registre d'état (SR : status register) et le PC (program counter).

3.5.1 Registres

La PCU comporte deux registres qui permettent de configurer et de consulter son état actuel :

- OMR (Operating mode register),
- SR (Status Register, cfr. fig. 3.7)

Le contrôle de l'exécution du code est effectué par quatre registres de la PCU :

- PC (Program Counter Register), contient l'adresse des mots d'instruction dans l'espace mémoire de programme
- LA (Loop Address Register), indique la position du dernier mot d'instruction dans une boucle matérielle
- LC (Loop Counter Register), spécifie le nombre de répétitions d'une boucle
- VBA (Vector Base Address Register), adresse de base de la table des vecteurs d'interruptions

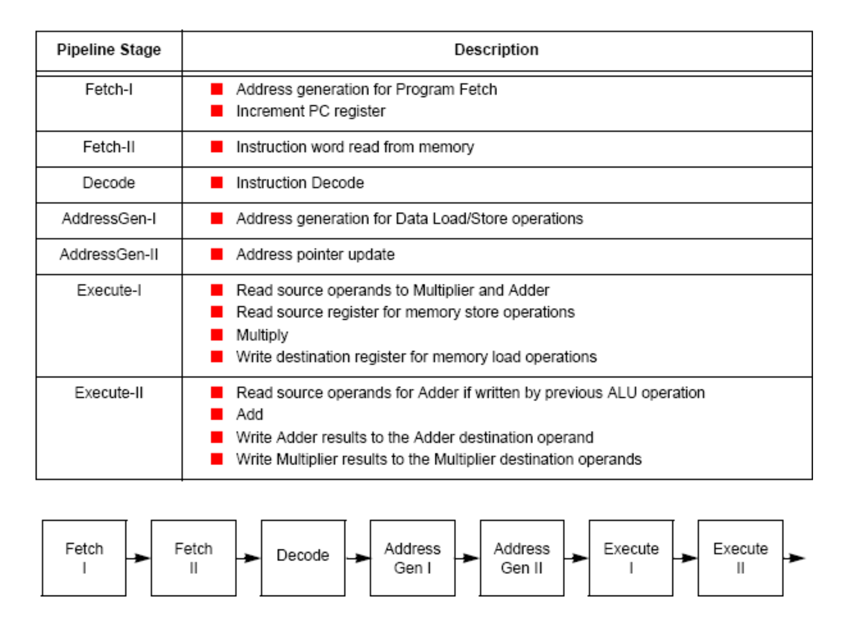
Extended Mode Register (EMR)								Mode Register (MR)								Condition Code Register (CCR)							
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP[1–0]		RM	SM	CE		SA	FV	LF	DM	SC		S[1–0]		I[1–0]		S	L	E	U	N	Z	V	C
Reset:																							
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Reserved bit. Read as zero; write to zero for future compatibility

FIG. 3.7 – Registre d'état (source : manuel de la famille 56300)

3.6 Pipeline

Le pipeline se compose des étages suivants (cfr. fig. 3.8) :
- deux étages d'extraction (fetch stage)
- un étage de décodage
- deux étages de génération d'adresse
- deux étages d'exécution



Pipeline Stage	Description
Fetch-I	Address generation for Program Fetch Increment PC register
Fetch-II	Instruction word read from memory
Decode	Instruction Decode
AddressGen-I	Address generation for Data Load/Store operations
AddressGen-II	Address pointer update
Execute-I	Read source operands to Multiplier and Adder Read source register for memory store operations Multiply Write destination register for memory load operations
Execute-II	Read source operands for Adder if written by previous ALU operation Add Write Adder results to the Adder destination operand Write Multiplier results to the Multiplier destination operands

FIG. 3.8 – Schéma fonctionnel du pipeline

3.6.1 Conflits

Dû à l'utilisaton d'un pipeline, il existe certaines restrictions sur les séquences d'opérations. On trouvera ci-dessous les conflits automatiquement résolus par le DSP. Il existe néanmoins des restrictions supplémentaires. Pour plus de renseignements, se référer au manuel d'utilisateur de la famille Motorola 56300.

3.6.1.1 Blocage arithmétique (Arithmetic Stall)

Puisque chaque instruction de la Data ALU s'effectue en 2 périodes d'horloge une condition d'interblocage se produit pendant une tentative de lecture d'un accumulateur lorsque l'instruction précédente est une instruction de Data ALU qui spécifie ce même accumulateur comme destination. Cette situation est détectée en hardware et un cycle d'attente est introduit entre les deux[2].

[2] Astuce de programmation : pour éviter ce délai, il est recommandé d'insérer une instruction utile (si possible).

3.6.1.2 Blocage d'état (Status Stall)

Ce blocage se produit lors d'une tentative de lecture du registre d'état alors que l'instruction précédente ou l'antépénultième est une instruction de Data ALU, ou une lecture d'accumulateur, mettant à jour les bits S et L du registre d'état. Un ou deux cycles d'attente sont insérés automatiquement au niveau hardware.

3.6.1.3 Blocage de transfert (Transfert Stall)

Le blocage de transfert se produit quand l'accumulateur source d'une instruction Data ALU est identique à l'accumulateur destination de l'instruction Data ALU précédente. Un cycle d'attente est inséré au niveau hardware.

Chapitre 4

Outils de développement

4.1 Compilateur et language assembleur

Les programmes développés pour le DSP Motorola 56311 dans le cadre de ce cours seront écrits en assembleur dans un fichier *.asm, à l'aide d'un éditeur de texte tel que le Notepad de Windows. La compilation du code est effectuée à l'aide du fichier "asm56300.exe".

L'assembleur Motorola traite le fichier *.asm en deux passages. Au premier passage, il construit les tables des symboles et des macros et, au second passage, le fichier objet est généré avec les références aux tables créées durant le premier passage et le listing du programme source est produit. Les erreurs détectées sont écrites dans ce listing.

Suite aux fonctionnalités avancées d'adressage (circulaire et reverse-carry), le langage assembleur fournit des directives pour établir l'adresse de base des buffers, leur allouer de la mémoire et initialiser leurs valeurs. Cependant, les buffers peuvent aussi être alloués manuellement.

4.1.1 Généralités

L'architecture du DSP 56311 possède les quatre espaces mémoire suivants : X, Y, L (concaténation de X et Y), P (programme). Puisque les transferts entre espaces mémoire sont autorisés, le code objet peut être chargé dans un emplacement mémoire et être transféré dans un autre emplacement mémoire avant l'exécution du programme (par ex. : un programme situé dans une EPROM externe est transféré dans une RAM externe avant l'exécution). Un transfert de ce type est appelé un "overlay". L'adresse et l'espace mémoire où le code est chargé sont l'espace mémoire de chargement (load memory space) et l'adresse de chargement (load address) tandis que l'adresse et l'espace mémoire où le code est exécuté sont l'espace mémoire d'exécution (**runtime** memory space) et l'adresse d'exécution (**runtime** address).

L'assembleur fonctionne soit dans le mode absolu, soit dans le mode relatif. En mode relatif, l'assembleur crée des fichiers objets translatables[1] (*.CLN) qui peuvent être combinés et translatés par l'éditeur de liens, tandis qu'en mode absolu[2], l'assembleur crée des fichiers objets absolus (*.CLD) qui ne peuvent pas être translatés mais qui peuvent être chargés directement pour exécution.

[1] Translatable : caractérise un programme en mesure d'occuper différentes places en mémoire au moment de son chargement et de son exécution.

[2] C'est le mode qui sera utilisé lors des laboratoires.

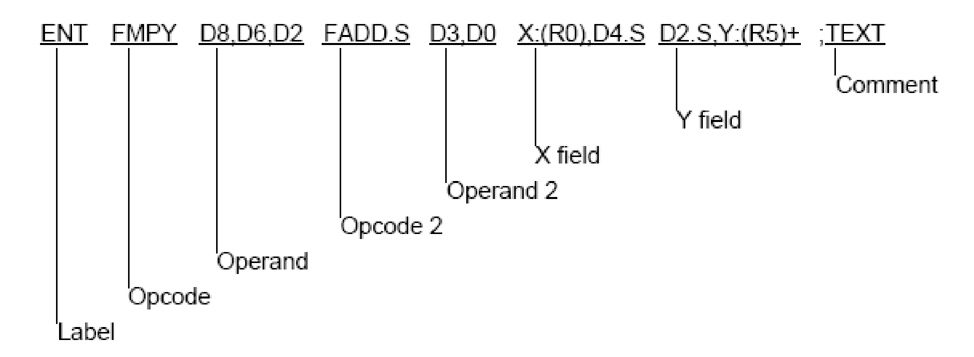


FIG. 4.1 – Schéma général d'une instruction en assembleur

4.1.2 Fichier source

Le language assembleur Motorola impose une certaine structure au code et il respecte la casse, sauf pour les mnémoniques d'instructions et les directives. Une instruction peut s'étendre sur plusieurs lignes (au moyen du symbole "\") et ne peut pas dépasser 255 caractères. En outre, le premier caractère d'un symbole doit être alphanumérique.

Ce language possède des opérateurs variés ainsi que des fonctions intégrées (ex : cosinus et sinus).

4.1.2.1 Structure d'une instruction

Le schéma général d'une instruction est illustré à la figure 4.1

- Le champ "Label" : un espace ou une tabulation comme premier caractère d'une ligne indique que le champ label est vide, un caractère alphabétique en premier indique qu'il existe un label.
- Le champ "Opcode" : il peut être de trois types. "Opcode"[3], "directive"[4] ou "macro call".
- Le champ "Operand" : Il doit être séparé du champ "Opcode" par un ou plusieurs espaces/tabulations et peut contenir un symbole, une expression ou une combinaison de symboles et d'expressions séparées par des virgules. Le format du champ operand pour une instruction particulière est spécifié dans le manuel d'utilisateur de la famille 56300.
- Les champs "Opcode 2" et "Operand 2" : Ne concernent pas la famille 56000.
- Les champs transferts de données "X" et "Y" : Spécifient les transferts de données s'exécutant en parallèle d'une opération. Chaque transfert de donnée est spécifié par ses deux opérandes séparées par une virgule (sans espace) et les transferts sont séparés par un ou plusieurs espaces/tabulations.
  Ex : L'instruction MAC x0,y0,a y :(r4),y1 va placer la valeur contenue à l'adresse de la mémoire Y spécifiée par le registre r4 dans le registre y1 en parallèle à l'exécution de l'opération MAC.
- Le champ "Comment" : Les commentaires commencent par un " ;".

ATTENTION, les opérateurs qui suivent ne concernent que le langage ASSEMBLEUR et ne constituent EN AUCUN CAS des instructions du DSP ! ! !

4.1.2.2 Les constantes numériques

Les constantes numériques peuvent être exprimées dans trois bases :

[3]Mnémoniques qui correspondent directement aux instructions machines du DSP.
[4]Code d'opérations spéciales connues par l'assembleur qui contrôle le processus d'assemblage.

- binaire : %11010
- hexadécimale : $12FF (ou $12ff)
- décimale : 12345 (entier), 6E10 (floating point), .6 (floating point), 2.7e2 (floating point).

4.1.2.3 Les opérateurs unaires

- + : retourne la valeur de son opérande
- - : retourne la valeur négative de son opérande
- ~ : complément à 1
- ! : négation logique

4.1.2.4 Les opérateurs arithmétiques

- + : addition
- - : soustraction
- * : multiplication
- / : division
- % : modulo

4.1.2.5 Opérateurs de décalage

- < < : décalage à gauche du nombre de bits spécifié par l'opérande de droite (seulement pour les entiers)
- > > : décalage à droite du nombre de bits spécifié par l'opérande de droite (seulement pour les entiers)

4.1.2.6 Opérateurs relationnels

- < : plus petit que.
- <= : plus petit ou égal à.
- > : plus grand que.
- >= : plus grand ou égal à.
- == : égal à.
- != : différent de.

4.1.2.7 Opérateurs de manipulation de bits (bitwise operators)

Ces instructions ne s'appliquent qu'aux entiers.
- & : ET
- | : OU
- ^ : XOR

4.1.2.8 Opérateurs logiques

- && : ET logique
- || : OU logique

4.1.3 Directives principales

INCLUDE inclusion de fichiers secondaires

ORG est utilisé pour spécifié des adresses et des changements d'espace mémoire (ex : ORG p :$100 sélectionne P comme étant l'espace mémoire d'exécution (runtime memory space : X, Y, L ou P) et initialise le compteur de position d'exécution à la valeur hexadécimale 100 en espace mémoire P).

EQU identifie un symbole à une valeur

BSC block storage of constant. L'assembleur alloue et initialise un bloc de mots. Ce bloc de mot commence à l'adresse donnée par la valeur du location counter dans l'espace mémoire courant. Le nombre de mots est spécifié par le premier argument et le deuxième spécifie la valeur d'initialisation (0 par défaut).

BSM block storage modulo. L'assembleur alloue et initialise un bloc de mots pour un buffer modulo. Le nombre de mots est spécifié par le premier argument et le deuxième spécifie la valeur d'initialisation (0 par défaut). Au début, le compteur de position est avancé à une adresse de base convenant pour l'adressage modulo (voir la sous-section 6.2.3). A la fin, le compteur de position est avancé du nombre de mots générés.

DC define constant. Alloue et initialise un mot en mémoire pour chaque argument (séparés par une virgule). Les arguments sont stockés à des adresses consécutives en commencant à l'adresse donnée par la valeur du location counter dans l'espace mémoire courant.

DS define storage. Réserve un bloc mémoire de taille égale au nombre de mots spécifié par l'argument. Ce bloc de mot commence à l'adresse donnée par la valeur du location counter dans l'espace mémoire courant.

DSM define modulo storage. Réserve, pour l'adressage modulo, un bloc mémoire de taille égale au nombre de mots spécifié par l'argument. Au début, le compteur de position d'exécution est avancé à une adresse de base convenant à l'adressage modulo. Puis, pour finir, il est avancé de la valeur entière donnée par l'argument.

4.1.4 Instructions assembleur principales

4.1.4.1 Principaux caractères significatifs

; : Délimiteur de commentaires

\ : Caractère de prolongation de ligne. Se place en bout de ligne.

*** :** Substitution du compteur de position (location counter). Quand l'astérisque est utilisée comme opérande dans une expression, elle représente la valeur entière courante du compteur de position d'exécution (runtime location counter).
Ex :
```
ORG    X :$100
XBASE      EQU    *+$20        ; donc, XBASE = $120
```

: Opérateur de mode d'adressage immédiat. Il indique à l'assembleur d'utiliser le mode d'adressage immédiat.

@ : Délimiteur de fonction. Le compilateur possède des fonctions intégrées qui commencent toutes par @. Par exemple, @SIN() calcule un sinus et @SQT calcule une racine carrée. **Attention, ces fonctions ne concernent que l'assembleur et pas le DSP.**

4.2 Débuggeur

A venir.

Chapitre 5

Instructions

5.1 Instructions de base

La plupart des instructions spécifient un mouvement de données sur le bus XDB et/ou YDB, ainsi qu'une opération Data ALU en un seul mot d'instruction. Le noyau du DSP 56300 effectue ces opérations en parallèle. Dans cette section, on trouvera les instructions les plus courantes, tandis que les sections suivantes fourniront une liste exhaustive des instructions de la famille 56300.

Déplacement (MOVE)

Operation	Assembler Syntax
$S \rightarrow D$	MOVE S,D

Déplace le contenu de la source S vers la destination D.

En plus de cette instruction de déplacement, on exploitera au maximum la possibilité de "move" parallèle qui accompagne 30 instructions sur les 62 disponibles. Cette possibilité permet à une opération de la Data ALU d'effectuer en parallèle à son exécution jusqu'à 2 déplacements sur les bus de données (1 par bus XDB et YDB).

Les différentes possibilités de déplacements sont résumées à la figure 5.1.

Remise à zéro (CLR)

Operation	Assembler Syntax
$0 \rightarrow D$ (parallel move)	CLR D (parallel move)

Met à 0 l'accumulateur (A ou B) destination D. C'est une instruction qui porte sur 56 bits.

Instruction	Description	Page
MOVE	Move Data	page 13-111
	No Parallel Data Move	page 13-112
I	Immediate Short Data Move	page 13-113
R	Register-to-Register Data Move	page 13-115
U	Address Register Update	page 13-117
X:	X Memory Data Move	page 13-118
X:R	X Memory and Register Data Move	page 13-120
Y	Y Memory Data Move	page 13-122
R:Y	Register and Y Memory Data Move	page 13-124
L:	Long Memory Data Move	page 13-126
X:Y:	X Y Memory Data Move	page 13-128

FIG. 5.1 – Instruction MOVE (source : manuel de la famille 56300)

Multiplication et accumulation (MAC)

Operation	Assembler Syntax
D ± S1 * S2 → D (parallel move)	MAC (±)S1,S2,D (parallel move)
D ± S1 * S2 → D (parallel move)	MAC (±)S2,S1,D (parallel move)
D ± (S1 * 2^{-n}) → D (no parallel move)	MAC (±)S,#n,D (no parallel move)

Instruction Fields

{S1,S2}	QQQ	Source registers S1,S2 [X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
{D}	d	Destination accumulator [A,B]
{±}	k	Sign [+,–]

{S}	QQ	Source register [Y1,X0,Y0,X1]
{D}	d	Destination accumulator [A,B]
{±}	k	Sign [+,–]
{#n}	ssss	Immediate operand

Multiplie les deux opérandes sources 24 bits S1 et S2 et additionne/soustrait le produit à/de l'accumulateur destination D.

Addition (ADD)

Operation		Assembler Syntax	
S + D → D	(parallel move)	ADD S,D	(parallel move)
#xx + D → D		ADD #xx,D	
#xxxx + D → D		ADD #xxxx,D	

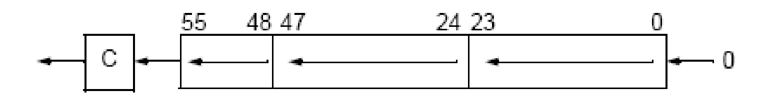
Instruction Fields

{S}	JJJ	Source register [B/A,X,Y,X0,Y0,X1,Y1]
{D}	d	Destination accumulator [A/B]
{#xx}	iiiiii	6-bit Immediate Short Data
{#xxxx}		24-bit Immediate Long Data extension word

Additionne l'opérande source S à l'opérande destination D et y enregistre le résultat.

Décalage à gauche (ASL)

Operation



Assembler Syntax

ASL D (parallel move)
ASL #ii,S2,D
ASL S1,S2,D

Instruction Fields

{S2}	S	Source accumulator [A,B]
{D}	D	Destination accumulator [A,B]
{S1}	sss	Control register [X0,X1,Y0,Y1,A1,B1]
{#ii}	iiiiii	6-bit unsigned integer [0–40] denoting the shift amount

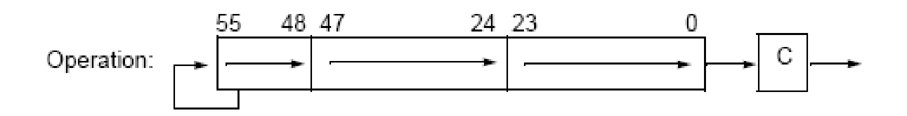
In the control register S1: bits 5–0 (LSB) are used as the #ii field, and the rest of the register is ignored.

Décalage simple : Décale arithmétiquement l'accumulateur destination d'un bit vers la gauche et y enregistre le résultat. Le MSB est déplacé dans le bit de "Carry" (C) et un "0" est placé dans le LSB.

Décalages multiples : Le contenu de l'accumulateur source S2 est décalé de #ii bits vers la gauche. Les bits décalés hors de la position 55 sont perdus, sauf le dernier qui est stocké dans le bit C. Les bits de droite laissés libres sont remplis par des 0, et le résultat est placé dans l'accumulateur destination. Le nombre de bits décalés est spécifié soit par les six bits dans l'instruction, soit par les six LSB de S1. Si un décalage de 0 est spécifié, le bit C est mis à 0. L'instruction ASL met le bit V à 1 si un overflow[1] se produit.

C'est une opération qui porte sur les 56 bits et qui, mathématiquement revient à multiplier la valeur du nombre par 2.

Décalage à droite (ASR)



Assembler Syntax

ASR D (parallel move)
ASR #ii, S2,D
ASR S1,S2,D

Instruction Fields

{S2}	S	Source accumulator [A,B]
{D}	D	Destination accumulator [A,B]
{S1}	sss	Control register [X0,X1,Y0,Y1,A1,B1]
{#ii}	iiiiii	6-bit unsigned integer [0–40] denoting the shift amount

In the control register S1: bits 5–0 (LSB) are used as the #ii field, and the rest of the register is ignored.

Décalage simple : Le LSB est décalé dans le bit C et le MSB est maintenu constant.

Décalages multiple : Les bits décalés hors de la position 0 sont perdus, sauf le dernier qui est placé dans le bit C. Les bits à gauche laissés libre par le décalage sont remplacés par la valeur du MSB. Si un décalage de 0 est spécifié, le bit C est mis à 0.

C'est une opération portant sur 56 ou 40 bits, en fonction du bit SA du registre d'état SR. Elle correspond, mathématiquement, à une division par 2.

Instruction vide (NOP)

Operation	Assembler Syntax
PC + 1 → PC	NOP

[1] Une condition d'overflow se produit lorsque le résultatde la Data ALU n'est pas représentable dans l'accumulateur destination (sauf cas particulier).

Incrémente le compteur de programme (PC) et les actions du pipeline en cours sont achevées. L'exécution se poursuit avec l'instruction suivante.

Instruction de boucle (DO)

Operation	Assembler Syntax
SP + 1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC SP + 1 → SP;PC → SSH;SR → SSL;expr – 1 → LA 1 → LF	DO [X or Y]:ea,expr
SP + 1 → SP;LA → SSH;LC → SSL;[X or Y]:aa → LC SP +1 → SP;PC → SSH;SR → SSL;expr – 1 → LA 1 → LF	DO [X or Y]:aa,expr
SP + 1 → SP;LA → SSH;LC → SSL;#xxx → LC SP+1 → SP;PC → SSH;SR → SSL;expr – 1 → LA 1 → LF	DO #xxx,expr
SP + 1 → SP;LA → SSH;LC → SSL;S → LC SP + 1 → SP;PC → SSH;SR → SSL;expr – 1 → LA 1 → LF	DO S,expr

End of Loop:
SSL(LF) → SR;SP – 1 → SP
SSH → LA;SSL → LC;SP – 1 → SP

Instruction Fields

{ea}	MMMRRR	Effective Address
{X/Y}	S	Memory Space [X,Y]
{expr}		24-bit Absolute Address in 16-bit extension word
{aa}	aaaaaa	Absolute Address [0–63]
{#xxx}	hhhhiiiiiiii	Immediate Short Data [0–4095]
{S}	DDDDDD	Source register [all on-chip registers, **except SSH**]

Répète les instructions comprises entre "DO" et le label "expr", un nombre de fois spécifié par l'opérande source. La gestion de la boucle est prise en charge par le DSP sans surcoût de cycles d'instructions. Les boucles DO peuvent être imbriquées mais ne peuvent pas avoir le même label de fin.

Saut vers une sous-routine (JSR)

Operation	Assembler Syntax
SP + 1 → SP; PC → SSH; SR → SSL; 0xxx → PC	JSR xxx
SP + 1 → SP; PC → SSH; SR → SSL; ea → PC	JSR ea

Instruction Fields

{xxx}	aaaaaaaaaaaa	Short Jump Address
{ea}	MMMRRR	Effective Address

Saute à la sous-routine dont la position est donnée par l'adresse effective de l'instruction (en pratique, un label).

Retour d'une sous-routine (RTS)

Operation	Assembler Syntax
SSH → PC; SP – 1 → SP	RTS

Instruction Fields None

Retourne de la sous-routine et continue à l'instruction suivant l'appel de la sous-routine.

5.2 Instructions arithmétiques

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
ABS	Absolute Value	√
ADC	Add Long With Carry	√
ADD	Add	√
ADD (imm.)	Add (immediate operand)	
ADDL	Shift Left and Add	√
ADDR	Shift Right and Add	√
ASL	Arithmetic Shift Left	√
ASL (mb.)	Arithmetic Shift Left (multi-bit)	
ASL (mb., imm.)	Arithmetic Shift Left (multi-bit, immediate operand)	
ASR	Arithmetic Shift Right	√
ASR (mb.)	Arithmetic Shift Right (multi-bit)	
ASR (mb., imm.)	Arithmetic Shift Right (multi-bit, immediate operand)	
CLR	Clear Accumulator	√
CMP	Compare	√
CMP (imm.)	Compare (immediate operand)	
CMPM	Compare Magnitude	√
CMPU	Compare Unsigned	
DEC	Decrement by One	
DIV	Divide Iteration	
DMAC	Double Precision Multiply-Accumulate With Right Shift	
INC	Increment by One	
MAC	Signed Multiply-Accumulate	√
MAC (su,uu)	Mixed Multiply-Accumulate	
MACI	Signed Multiply-Accumulate With Immediate Operand	
MACR	Signed Multiply-Accumulate and Round	√
MACRI	Signed Multiply-Accumulate and Round With Immediate Operand	
MAX	Transfer by Signed Value	√
MAXM	Transfer by Magnitude	√

MPY	Signed Multiply	√
MPY (su,uu)	Mixed Multiply	
MPYI	Signed Multiply With Immediate Operand	
MPYR	Signed Multiply and Round	√
MPYRI	Signed Multiply and Round With Immediate Operand	
NEG	Negate Accumulator	√
NORM	Norm Accumulator Iteration	
NORMF	Fast Accumulator Normalization	
RND	Round Accumulator	√
SBC	Subtract Long With Carry	√
SUB	Subtract	√
SUB (imm.)	Subtract (immediate operand)	
SUBL	Shift Left and Subtract Accumulators	√
SUBR	Shift Right and Subtract Accumulators	√
Tcc	Transfer Conditionally	
TFR	Transfer Data ALU Register	√
TST	Test Accumulator	√

5.3 Instructions logiques

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
AND	Logical AND	√
AND (imm.)	Logical AND (immediate operand)	
ANDI	AND Immediate to Control Register	
CLB	Count Leading Bits	
EOR	Logical Exclusive OR	√
EOR (imm.)	Logical Exclusive OR (immediate operand)	
EXTRACT	Extract Bit Field	
EXTRACT (imm.)	Extract Bit Field (immediate operand)	
EXTRACTU	Extract Unsigned Bit Field	
EXTRACTU (imm.)	Extract Unsigned Bit Field (immediate operand)	
INSERT	INSERT Bit Field	
INSERT (imm.)	INSERT Bit Field (immediate operand)	
LSL	Logical Shift Left	√
LSL (mb.)	Logical Shift Left (multi-bit)	
LSL (mb., imm.)	Logical Shift Left (multi-bit, immediate operand)	
LSR	Logical Shift Right	√
LSR (mb.)	Logical Shift Right (multi-bit)	
LSR (mb.,imm.)	Logical Shift Right (multi-bit, immediate operand)	
MERGE	Merge Two Half Words	
NOT	Logical Complement	√
OR	Logical Inclusive OR	√
OR (imm.)	Logical Inclusive OR (immediate operand)	
ORI	OR Immediate With Control Register	
ROL	Rotate Left	√
ROR	Rotate Right	√

5.4 Instructions de manipulation de bits

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
BCHG	Bit Test and Change	
BCLR	Bit Test and Clear	
BSET	Bit Test and Set	
BTST	Bit Test	

5.5 Instructions de boucle

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
BRKcc	Conditionally Break the current Hardware Loop	
DO	Start Hardware Loop	
DO FOREVER	Start Infinite Loop	
DOR	Start PC-Relative Hardware Loop	
DOR FOREVER	Start PC-Relative Infinite Loop	
ENDDO	End Current DO Loop	

5.6 Instructions de déplacement

Mnemonic	Description	Parallel Instruction
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
LUA	Load Updated Address	
LRA	Load PC-Relative Address	
MOVE	Move Data Register	√
	No Parallel Data Move	
I	Immediate Short Data Move	√
R	Register-to-Register Data Move	√
U	Address Register Update	√
X:	X Memory Data Move	√
X:R	X Memory and Register Data Move	√
Y	Y Memory Data Move	√
R:Y	Register and Y Memory Data Move	√
L:	Long Memory Data Move	√
X:Y:	X Y Memory Data Move	√
MOVEC	Move Control Register	
MOVEM	Move Program Memory	
MOVEP	Move Peripheral Data	
VSL	Viterbi Shift Left	

5.7 Instructions de contrôle du programme

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
Bcc	Branch Conditionally	
BRA	Branch Always	
BRCLR	Branch if Bit Clear	
BRSET	Branch if Bit Set	
BScc	Branch to Subroutine Conditionally	
BSCLR	Branch to Subroutine if Bit Clear	
BSR	Branch to Subroutine	
BSSET	Branch to Subroutine if Bit Set	
DEBUG	Enter Debug Mode	
DEBUGcc	Enter Debug Mode Conditionally	
IFcc	Execute Conditionally Without CCR Update	
IFcc.U	Execute Conditionally and Update CCR	
ILLEGAL	Illegal Instruction Interrupt	
Jcc	Jump Conditionally	
JCLR	Jump if Bit Clear	
JMP	Jump	
JScc	Jump to Subroutine Conditionally	
JSCLR	Jump to Subroutine if Bit Clear	
JSET	Jump if Bit Set	
JSR	Jump to Subroutine	
JSSET	Jump to Subroutine if Bit Set	
NOP	No Operation	
REP	Repeat Next Instruction	
RESET	Reset On-Chip Peripheral Devices	
RTI	Return From Interrupt	
RTS	Return From Subroutine	
STOP	Stop Instruction Processing	
TRAP	Software Interrupt	
TRAPcc	Conditional Software Interrupt	
WAIT	Wait for Interrupt or DMA Request	

5.8 Instructions de contrôle du cache d'instructions

Pour mémoire, il existe une possibilité de cache pour les instructions lues sur une mémoire de programme externe et qui reviennent fréquemment afin d'accélérer l'exécution du programme.

5.9 Exemple de programmation

A venir

Chapitre 6

Adressage

6.1 Modes d'adressage

Le noyau de la famille 56300 fournit quatre modes d'adressage. Ces quatre modes peuvent être utilisés aussi bien avec la mémoire interne qu'avec de la mémoire externe accessible par le port A.

6.1.1 Register Direct

L'opérande est contenue dans un (ou plusieurs) des registres de la Data ALU, des registres d'adresse ou des sept registres de contrôle.

Data or Control Register Direct : L'opérande est dans un, deux ou trois registres de la Data ALU

Address Register Direct : L'opérande est dans un des 24 registres d'adresse spécifié par une adresse effective dans l'instruction

6.1.2 Address Register Indirect

Le registre d'adresse pointe vers une adresse mémoire. L'adressage est dit "indirect" car le registre ne contient pas l'opérande mais son adresse (exemple type : MOVE x :(Rn), x0 ; la valeur contenue dans le registre Rn fournit l'adresse dans l'espace mémoire X où est enregistrée la valeur à déplacer vers x0). Ce type d'adressage permet de parcourir aisément des ensembles de données ordonnées (buffers, ...). Dans une grande partie des calculs suivants, l'arithmétique utilisée pour le calcul d'adresse est défini par la valeur du registre Mn.

- *No Update (Rn)*—The operand address is in the address register. The contents of the address register are unchanged by executing the instruction.

  `Example: MOVE x:(Rn),x0`

- *Post-Increment By One (Rn) +* —The operand address is in the address register. After the operand address is used, it is incremented by one and stored in the same address register. The Nn register is ignored.

  `Example: MOVE x:(Rn)+,x0`

- *Post-Decrement By One (Rn) –* —The operand address is in the address register. After the operand address is used, it is decremented by one and stored in the same address register. The Nn register is ignored.

  `Example: MOVE x:(Rn)-,x0`

- *Post-Increment By Offset Nn (Rn) + Nn*—The operand address is in the address register. After the operand address is used, it is incremented by the contents of the Nn register and stored in the same address register. The contents of the Nn register are unchanged.

  `Example: MOVE x:(Rn)+Nn,x0`

- *Post-Decrement By Offset Nn (Rn) – Nn*—The operand address is in the address register. After the operand address is used, it is decremented by the contents of the Nn register and stored in the same address register. The contents of the Nn register are unchanged.

  `Example: MOVE x:(Rn)-Nn,x0`

- *Indexed By Offset Nn (Rn + Nn)*—The operand address is the sum of the contents of the address register and the contents of the address offset register, Nn. The contents of the Rn and Nn registers are unchanged.

  `Example: MOVE x:(Rn+Nn),x0`

- *Pre-Decrement By One -(Rn)*—The operand address is the contents of the address register decremented by one. The contents of Rn are decremented by one and stored in the same address register before the memory access. The Nn register is ignored.

  `Example: MOVE x:-(Rn),x0`

- *Short Displacement (Rn + Short Displacement)*—The operand address is the sum of the contents of the address register Rn and a short signed displacement occupying seven bits in the instruction word. The displacement is first sign-extended to 24 bits (16 bits in SC mode) and then added to Rn to obtain the operand address. The contents of the Rn register are unchanged. The Nn register is ignored. This reference is classified as a memory reference.

  `Example: MOVE x:(Rn+63),x0`

- *Long Displacement (Rn + Long Displacement)*—This addressing mode requires one word (label) of instruction extension. The operand address is the sum of the contents of the address register and the extension word. The contents of the address register are unchanged. The Nn register is ignored. This reference is classified as a memory reference.

  `Example: MOVE x:(Rn+64),x0`

6.1.3 PC-relative

(Pour mémoire). L'adresse est obtenue par addition d'un décalage à la valeur du PC (program counter).

6.2 Arithmétique d'adressage

L'ALU d'adressage supporte l'adressage linéaire, modulo, multiple wrap-around modulo et reverse-carry pour tous les modes d'adressage du type "Address Register Indirect". La valeur à placer dans un des registres "Address Modifier" Mn, pour spécifier l'arithmétique à utiliser, est donnée dans la table ci-dessous.

Modifier Mn	Address Calculation Arithmetic
$XX0000	Reverse-Carry (Bit-Reverse)
$XX0001	Modulo 2
$XX0002	Modulo 3
:	:
$XX7FFE	Modulo 32767 (2^{15}-1)
$XX7FFF	Modulo 32768 (2^{15})
$XX8001	Multiple Wrap-Around Modulo 2
$XX8003	Multiple Wrap-Around Modulo 4
$XX8007	Multiple Wrap-Around Modulo 8
:	:
$XX9FFF	Multiple Wrap-Around Modulo 2^{13}
$XXBFFF	Multiple Wrap-Around Modulo 2^{14}
$XXFFFF	Linear (Modulo 2^{24})
Notes: 1. All other combinations are reserved. 2. XX can be any value.	

6.2.1 Adressage linéaire (M_n= $XXFFFF)

Adressage de type général. La modification d'adresse est effectuée linéairement sur les valeurs 24 bits.

6.2.2 Adressage reverse-carry

Utile pour l'implémentation de la FFT comportant 2^kpoints.

6.2.3 Adressage circulaire (modulo)

L'arithmétique modulo M impose à la valeur contenue dans Rn à rester dans une plage d'adresse de taille M, si la valeur M-1 est stockée dans Mn. Très utile pour la création de buffers circulaires utilisés dans les files FIFO, les buffers et les lignes à retards.

Il faut cependant savoir qu'il existe des contraintes sur l'emplacement mémoire de l'adresse de base de telles zones. Pour de plus amples détails, se référer au manuel de la famille 56300.

Si un offset Nn est utilisé, il faut $|Nn| \leq M$ sinon le résultat est imprédictible, sauf dans un cas particulier utilisé pour traiter séquentiellement des tableaux à N dimensions (pour mémoire).

6.2.4 Adressage circulaire à enroulements multiples (modulo multiple wrap-around)

L'adressage modulo multiple wrap-around est une variante de l'adressage circulaire. Le principe est identique mais la valeur du modulo M est obligatoirement une puissance de 2 comprise entre 2^1et 2^{14}. On peut, ici, utiliser des valeurs de Nn $>$ M.

6.3 Exemple de programmation

A venir.

Chapitre 7

DSP 56311

7.1 Présentation générale

Le DSP 56311 offre des performances de 150 millions d'instructions par secondes (MIPS). Il est doté, entre autre, d'un module EFCOP qui permet d'exécuter un filtrage en parallèle au noyau, d'un module DMA (Direct Memory Access), de 34 General-purpose In/Out (GPIO), de 4 lignes d'interruptions, de trois timers, d'un port parallèle (HI08 Host Interface), de deux ESSI (Enhanced Synchronous Serial Interface) et d'une SCI (Serial Communication Interface). Il dispose, en outre, d'une mémoire interne répartie entre la mémoire de programme, la mémoire de données X et la mémoire de données Y.

7.2 Périphériques

Les périphériques d'entrée/sortie sont controlés au moyen de registres mappés dans les 128 mots supérieurs de la mémoire de données X.

GPIO

Le port GPIO est constitué de 34 signaux bidirectionnels qui peuvent être configurés en signaux d'usage général ou en signaux dédiés aux périphériques. Son mode de fonctionnement est sélectionné dans le registre de contrôle.

Les signaux sont répartis en 5 groupes :

- Port B (partagés avec les signaux HI08)
- Port C (partagés avec les signaux ESSI0)
- Port D (partagés avec les signaux ESSI1)
- Port E (partagés avec les signaux SCI)
- Timers (partagés avec les signaux du timer triple)

HI08

L'HI08 est un port parallèle full-duplex à double buffer de largeur 8 bits. Il supporte de nombreux bus et fournit une connection facile ("glueless") avec des microprocesseurs, microordinateurs et des DSPs aux standards industriels.

ESSI

L'ESSI fournit un port série full-duplex pour la communication avec un codec, un DSP, un microprocesseur et des périphériques. L'interface est synchrone car tous les transferts série sont synchronisés sur une horloge.

SCI

Le SCI fournit un port série full-duplex pour la communication avec d'autres DSP, microprocesseurs ou périphériques tels que les modems. Il accepte tant les débits asynchrones standards de l'industrie que les communications synchrones rapides. C'est ce port qui est utilisé pour la communication série avec un PC.

7.3 Configuration de la mémoire

Le DSP 56311 peut adresser trois sets de 16M x 24 bits de mémoire (programme, données X et données Y). Ces trois espaces de mémoires sont tant internes qu'externes. De plus, il existe plusieurs mode de répartition de la mémoire interne entre la partie programme, données X et données Y.

Mémoire de programme

Elle est composée des parties suivantes : mémoire de programme interne, cache d'instruction (optionnel), ROM d'amorce de programme (bootstrap), expansion optionnelle de mémoire en externe.

Mémoire de données X

Elle est composée des parties suivantes : mémoire interne de données, espace interne réservé aux entrées/sorties (accès entre autre par instructions MOVE, MOVEP), expansion optionnelle de mémoire en externe.

Mémoire de données Y

Elle est composée des parties suivantes : mémoire interne de données, espace interne réservé aux entrées/sorties, expace externe résevé au entrées/sorties, expansion optionnelle de mémoire en externe.

Cartographie de la mémoire

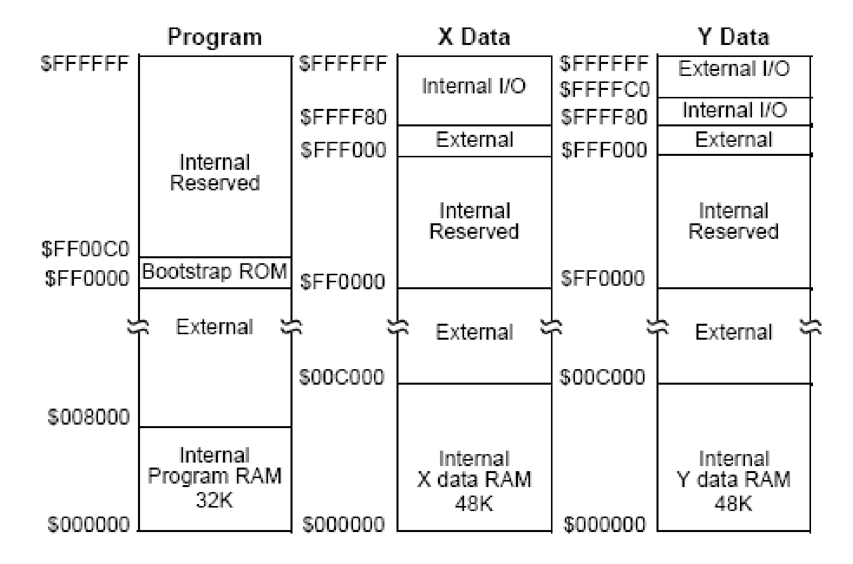
La configuration de l'espace mémoire et de la RAM est définie par les valeurs des bits MS (bit 7 du registre OMR), MSW[1 :0] (bits 22 et 21 du registre OMR), CE (bit 19 du registre SR) et SC (bit 13 du registre SR). Le schéma de l'organisation par défaut de la mémoire est donné par la figure 7.1.

7.4 Interface de communication série

A venir

7.5 Interruptions

A venir



Bit Settings				Memory Configuration				
MS	MSW [1:0]	CE	SC	Program RAM	X Data RAM*	Y Data RAM*	Cache	Addressable Memory Size
0	any value	0	0	32K $0000 – $7FFF	48K $0000 – $BFFF	48K $0000 – $BFFF	None	16 M

* Lowest 10K of X data RAM and 10K of Y data RAM are shared memory that can be accessed by the core and the EFCOP but not by the DMA controller.

FIG. 7.1 – Cartographie de l'organisation mémoire par défaut (source : manuel d'utilisateur du DSP 56311)

Chapitre 8

Carte d'évaluation EVM56311

Comme observé sur la figure 8.1, la carte EVM56311, utilisée aux laboratoires, est une carte d'évaluation qui comporte divers périphériques d'usage courant en plus d'un DSP 56311.

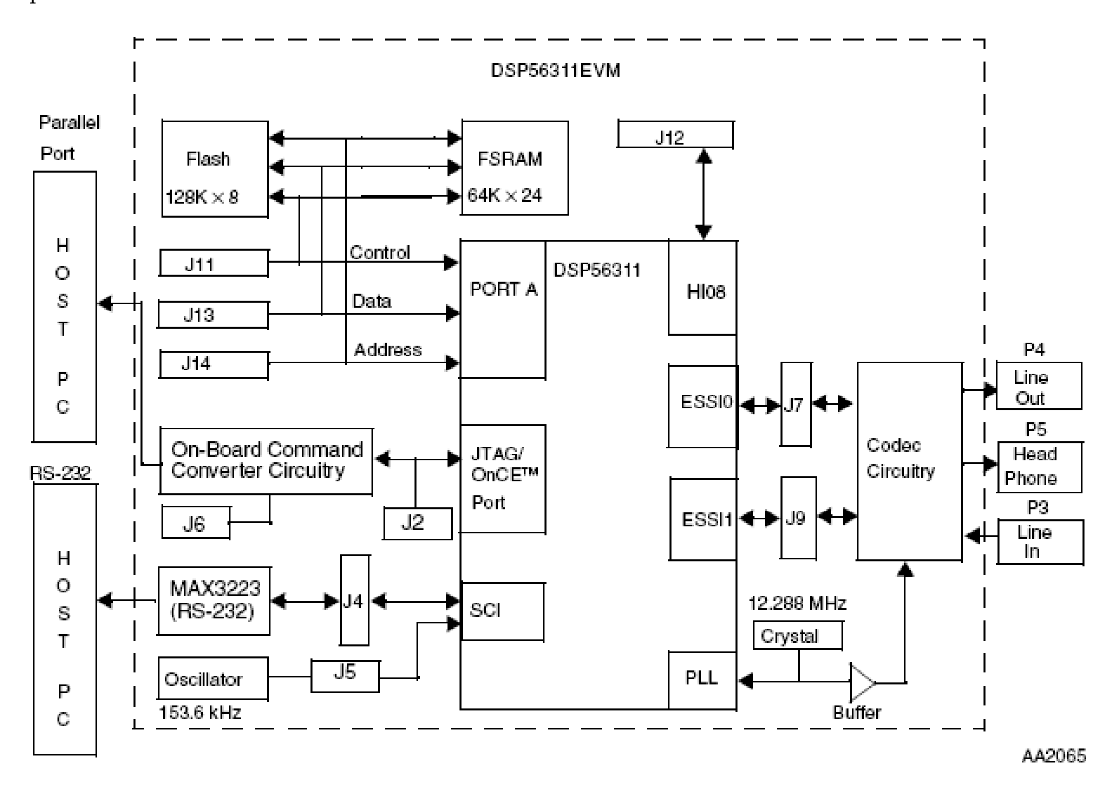


Fig. 8.1 – Schéma bloc fonctionnel de la carte 56311EVM (source : manuel de la carte d'évaluation 56311EVM)

On peut y trouver

- un codec audio (CS4218) connecté aux ports ESSI0 et ESSI1,
- une puce MAX3223 connectée au port SCI qui permet une communication série avec un PC (par ex.),
- un port parallèle connecté au port JTAG/OnCE qui assure la communication avec le débuggeur installé sur les ordinateurs de développement.
- de la mémoire RAM connectée au portA, qui permet d'étendre la mémoire interne du DSP,

- de la mémoire FLASH qui permet de faire fonctionner la carte d'évaluation en système indépendant.

Le CS4218 est un codec stéréo16 bits qui autorise des fréquences d'échantillonnage comprises entre 8 kHz et 48 kHz. Ces fréquences se règlent au moyen du jumper J8.

8.1 Interfaçage avec le codec CS4218

Pour l'utilisation du codec CS4218, la carte d'évaluation DSP56311EVM est fournie avec les fichiers sources suivants :

- Ioequ.asm : Associations ("equates") des entrées-sorties pour les modules de la carte d'évaluation
- Intequ.asm : Associations des interruptions pour les modules de la carte d'évaluation
- Ada_equ.asm : Associations nécessaires à l'initialisation du codec
- Ada_init.asm : Code d'initialisation pour l'ESSI et le codec
- Vectors.asm : Table de vecteurs d'interruptions pour les modules de la carte d'évaluation

Ces fichiers seront à inclure (commande include) dans les fichiers source utilisés aux laboratoires.

Le codec stéreo CS4218 comprend : deux convertisseurs A/D delta-sigma, deux convertisseurs D/A delta-sigma, des filtres d'entrée anti-alias, des filtres anti image en sortie, un gain d'entrée et une atténuation de sortie programmable. Le codec est connecté au DSP par 2 ports ESSI (ESSI0 pour le transfert de données et ESSI1 pour le transfert des informations de contrôle du codec). La programmation du codec comporte 3 étapes : la mise en place de constantes globales, l'interfaçage et l'initialisation de l'ESSI et du codec, l'inclusion de mécanismes de transfert de données (interruption dans ce cas-ci). Pour de plus amples détails, on consultera la note d'application AN1790 "Programming the CS4218 CODEC for Use With DSP56300 Devices".

Chapitre 9

Annexes

9.1 Branchements de la carte d'évaluation dans la chaine de mesures

A venir.

9.2 Utilisation de l'Audio Precision

A venir.